National Tsing Hua University
Fall 2023 11210IPT 553000
Deep Learning in Biomedical Optical Imaging
Homework 2

AUTHOR ONE[1] 張皓旻

Student ID:112022533

1. Task A: Performance between BCE loss and BC loss

1.1 Model 與 Hyperparameters 比較:

```
model1 = nn.Sequential(
        nn.Flatten(),

        nn.Linear(256*256*1,  512),
        nn.BatchNorm1d(512),
        nn.ReLU(),
        nn.Dropout(0.9),

        nn.Linear(512,  256),
        nn.BatchNorm1d(256),
        nn.ReLU(),
        nn.Dropout(0.9),

        nn.Linear(256,  256),
        nn.BatchNorm1d(256),
        nn.ReLU(),
        nn.Dropout(0.8),

        nn.Linear(256,  1)
).cuda()
```

Fig1.BCE model

```
model1 = nn.Sequential(
        nn.Flatten(),

        nn.Linear(256*256*1,  512),
        nn.BatchNorm1d(512),
        nn.ReLU(),
        nn.Dropout(0.9),

        nn.Linear(512,  256),
        nn.BatchNorm1d(256),
        nn.ReLU(),
        nn.Dropout(0.9),

        nn.Linear(256,  256),
        nn.BatchNorm1d(256),
        nn.ReLU(),
        nn.Dropout(0.8),

        nn.Linear(256,  2)
).cuda()
```

Fig2.CE model

Fig1 和 Fig2 分別是 BCE loss 所使用的 model 與 CE loss 所使用的 model，其區別為最後一層，BCE loss model 的輸出為一個，CE loss model 的輸出為兩個，而其中都採用了高機率的 Dropout 來避免 overfitting，並且兩種訓練方式都採用相同的 Hyperparameters 如以下 Table(1)。

Table(1) : 兩種不同 loss function 所使用的 Hyperparameters

Loss function	Batch size	Number of epochs	Optimizer	Learning rate scheduler
BCEWithLogitsLoss	32	40	Adam(model.parameters(), lr=1e-3)	CosineAnnealingLR
CrossEntropyLoss	32	40	Adam(model.parameters(), lr=1e-3)	CosineAnnealingLR

1.2 結果:

以下比較兩種 loss function 所訓練出來的 Accuracy 與 Loss，首先 Fig3 是 BCE loss 下的模型性能，其最佳的 Accuracy 為 Epochs = 39 那一輪，Train Accuracy 為 94.69%、 Loss 為 0.1536，Validation Accuracy 為 95.00%、Loss 為 0.1410，overfitting 的情況沒有發生，但 Validation Accuracy 波動較大。

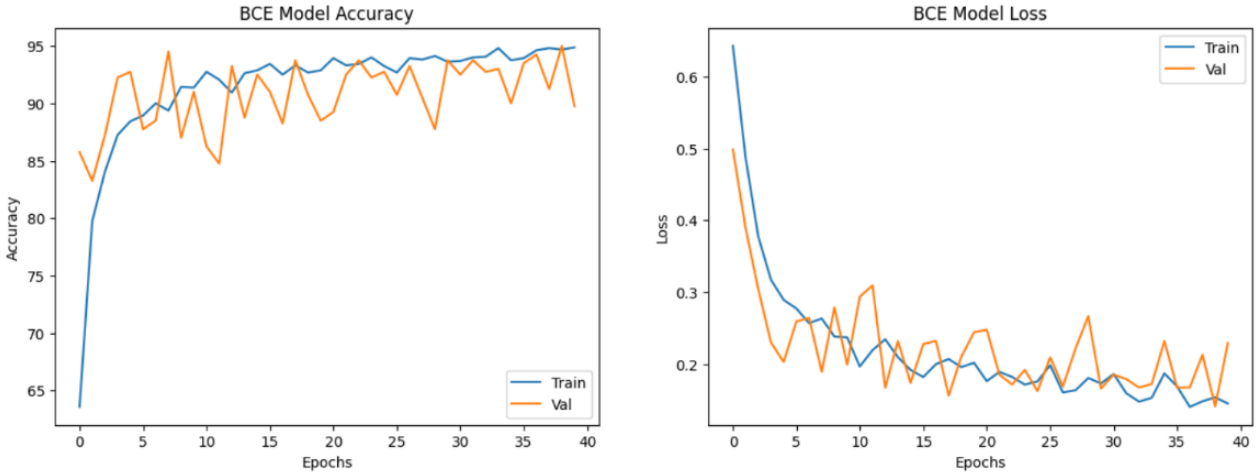

Fig3. BCE loss 下的模型性能

接下來是 CE loss 下的模型性能 Fig4，其最後一輪的 Train Accuracy 為 94.00%、 Loss 為 0.1684，Validation Accuracy 為 93.50%、Loss 為 0.1461，overfitting 的情況沒有發生，且 Validation Accuracy 波動較小，整體更為穩定。

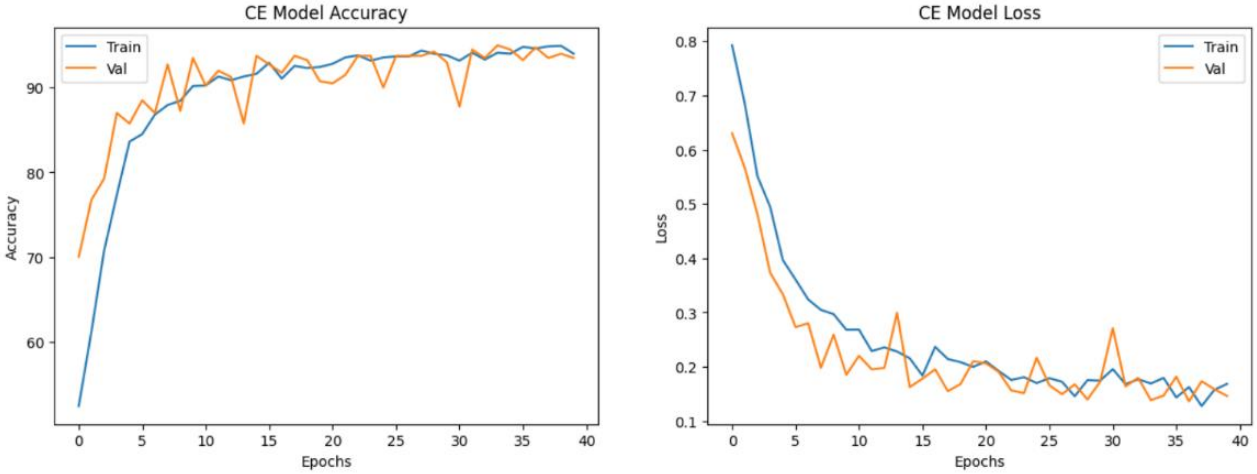

Fig4. CE loss 下的模型性能

之後我將兩種情況下的模型性能畫進同一張圖進行比較，如 Fig5，可以明顯的發現 BCE loss 的訓練收斂速度更快，推測是因為 BCE 更適合用於二元分類的關係，但之後 CE loss 的穩定度更好，推測可以更改 Learning rate scheduler 使 BCE loss 的最終值可以趨於穩定。

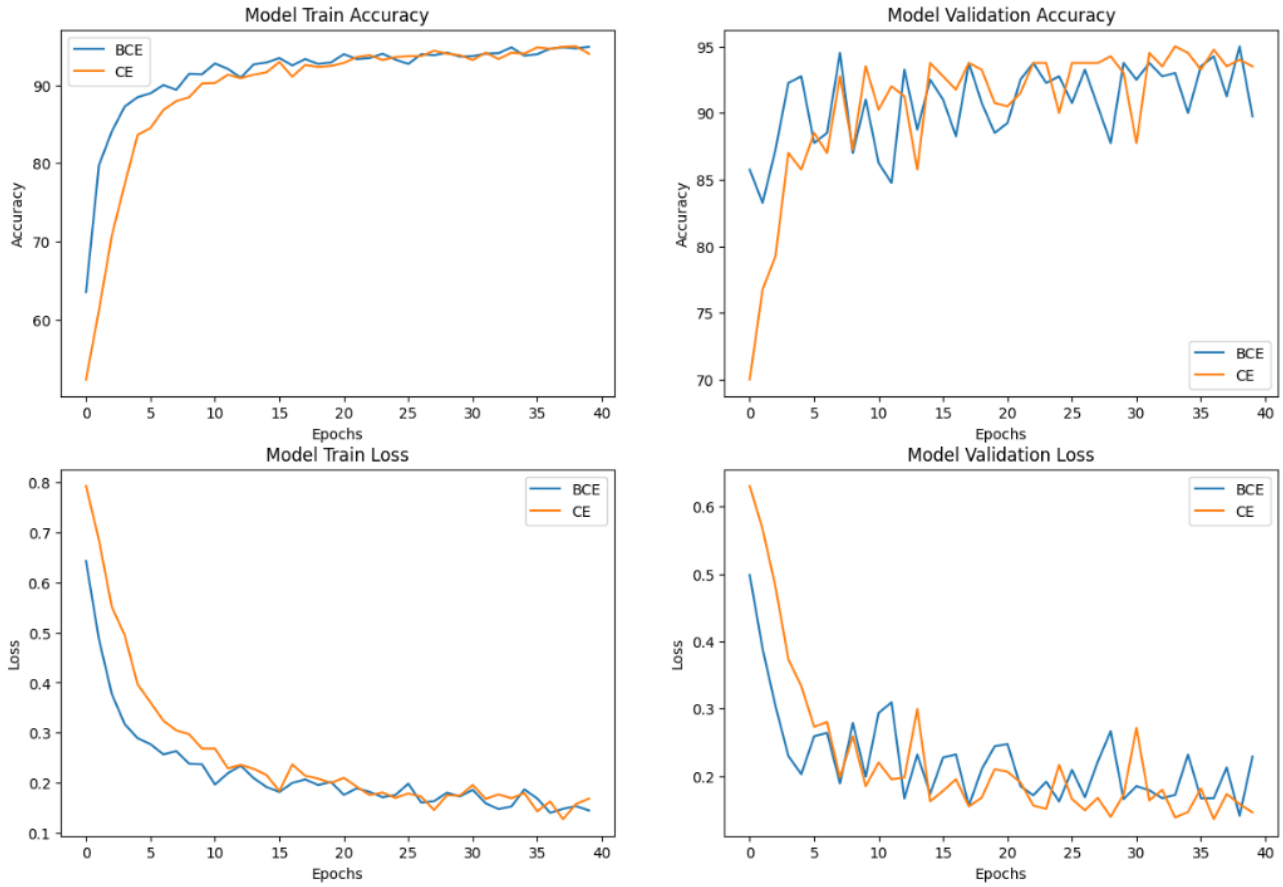

Fig5. BCE loss 與 CE loss 的比較

2. Task B: Performance between Different Hyperparameters

首先我想改變不同的 Learning rate scheduler，看看是否如我上面推測的一樣，可以使 BCE loss 下的 Accuracy 波動降低，分別比較 CosineAnnealingLR、StepLR 與 PolynomialLR 的差別，並且使用的 model 如下圖 Fig6 所示，其他的 Hyperparameters 維持一樣，如 Table(2)所示。

```
model1 = nn.Sequential(
        nn.Flatten(),

        nn.Linear(256*256*1,  512),
        nn.BatchNorm1d(512),
        nn.ReLU(),
        nn.Dropout(0.9),

        nn.Linear(512,  256),
        nn.BatchNorm1d(256),
        nn.ReLU(),
        nn.Dropout(0.9),

        nn.Linear(256,  256),
        nn.BatchNorm1d(256),
        nn.ReLU(),
        nn.Dropout(0.8),

        nn.Linear(256,  1)
).cuda()
```

Fig6. 本次實驗所使用之 model

Learning rate scheduler	Batch size	Number of epochs	Optimizer	Loss function
CosineAnnealingLR	32	40	Adam(model.parameters(), lr=1e-3)	BCEWithLogitsLoss
StepLR	32	40	Adam(model.parameters(), lr=1e-3)	BCEWithLogitsLoss
PolynomialLR (Power1)	32	40	Adam(model.parameters(), lr=1e-3)	BCEWithLogitsLoss

Table(2)：本次實驗所使用的 Hyperparameters

結果:

如下圖 Fig7、Fig8、Fig9，分別代表不同 Learning rate scheduler 下的 Accuracy 表現，用了 StepLR 的 Accuracy 約 90%，表現比另外兩者差，CosineAnnealingLR 與 PolynomialLR 的 Accuracy 約 94%。

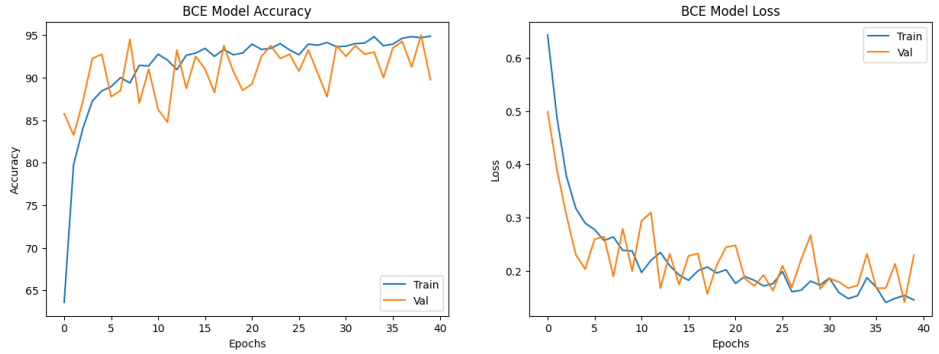

Fig7. CosineAnnealingLR 模式的 model Accuracy

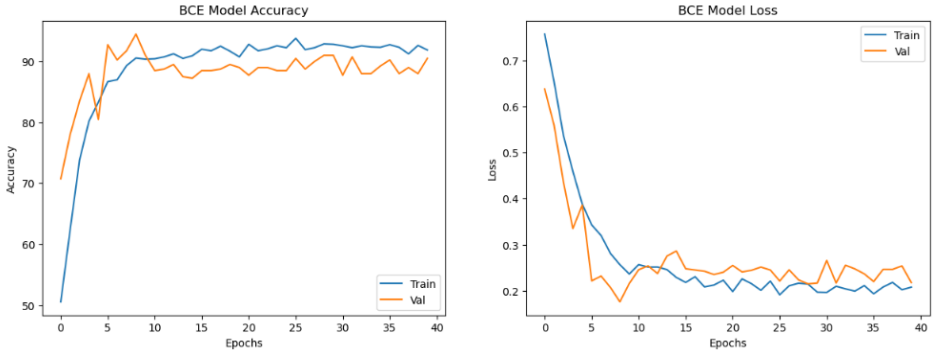

Fig8. StepLR 模式的 model Accuracy

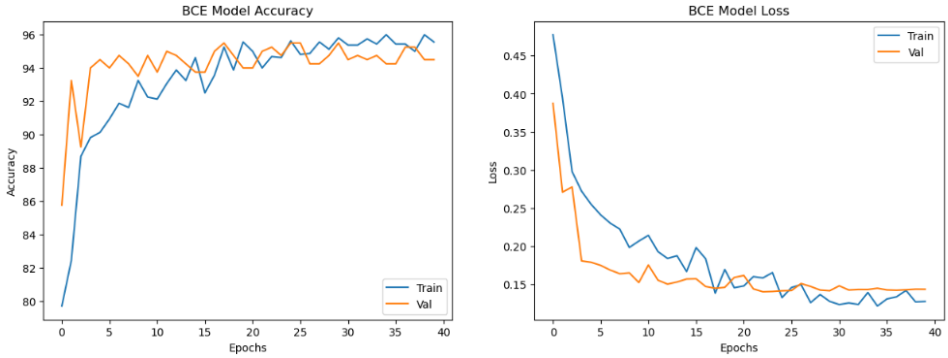

Fig9. PolynomialLR (Power1) 模式的 model Accuracy

接下來我想要改變的是 model 的激活函數，比較 ReLU、 LeakyReLU 與 Sigmoid 之間的區別，model 分別如下圖 Fig10、Fig11、Fig12，其餘 Hyperparameters 使用與 Table(2)的 CosineAnnealingLR 的相同。

```
model = nn.Sequential(
    nn.Flatten(),

    nn.Linear(256*256*1, 512),
    nn.BatchNorm1d(512),
    nn.ReLU(),
#     nn.LeakyReLU(),
#     nn.Sigmoid()
    nn.Dropout(0.9),

    nn.Linear(512, 256),
    nn.BatchNorm1d(256),
    nn.ReLU(),
#     nn.LeakyReLU(),
#     nn.Sigmoid()
    nn.Dropout(0.9),

    nn.Linear(256, 256),
    nn.BatchNorm1d(256),
    nn.ReLU(),
#     nn.LeakyReLU(),
#     nn.Sigmoid()
    nn.Dropout(0.8),

    nn.Linear(256, 1)
).cuda()
```

Fig10 .ReLU 函數 model

```
model = nn.Sequential(
    nn.Flatten(),

    nn.Linear(256*256*1, 512),
    nn.BatchNorm1d(512),
#     nn.ReLU(),
    nn.LeakyReLU(),
#     nn.Sigmoid()
    nn.Dropout(0.9),

    nn.Linear(512, 256),
    nn.BatchNorm1d(256),
#     nn.ReLU(),
    nn.LeakyReLU(),
#     nn.Sigmoid()
    nn.Dropout(0.9),

    nn.Linear(256, 256),
    nn.BatchNorm1d(256),
#     nn.ReLU(),
    nn.LeakyReLU(),
#     nn.Sigmoid()
    nn.Dropout(0.8),

    nn.Linear(256, 1)
).cuda()
```

Fig11 .LeakyReLU 函數 model

```
model = nn.Sequential(
    nn.Flatten(),

    nn.Linear(256*256*1, 512),
    nn.BatchNorm1d(512),
#     nn.ReLU(),
#     nn.LeakyReLU(),
    nn.Sigmoid()
    nn.Dropout(0.9),

    nn.Linear(512, 256),
    nn.BatchNorm1d(256),
#     nn.ReLU(),
#     nn.LeakyReLU(),
    nn.Sigmoid()
    nn.Dropout(0.9),

    nn.Linear(256, 256),
    nn.BatchNorm1d(256),
#     nn.ReLU(),
#     nn.LeakyReLU(),
    nn.Sigmoid()
    nn.Dropout(0.8),

    nn.Linear(256, 1)
).cuda()
```

Fig12 .Sigmoid 函數 model

結果:

如下圖 Fig13、Fig14、Fig15，分別代表不同激活函數下的 Accuracy 表現，使用 LeakyReLU 的 model 在 Accuracy 波動上比 ReLU 平緩，而 Sigmoid 函數的 model 收斂的非常緩慢，感覺效能遠不及另外兩個。

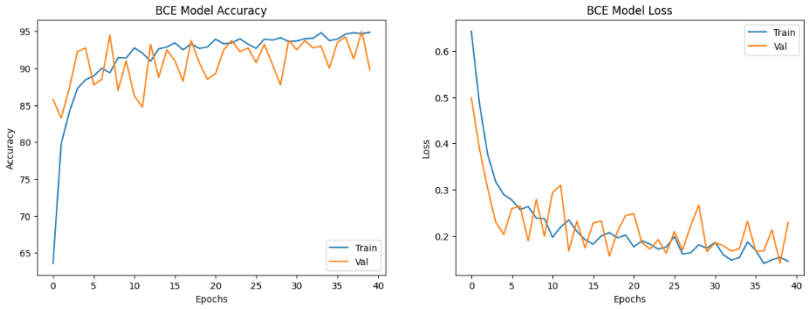

Fig13.使用 ReLU 函數的 model 之表現

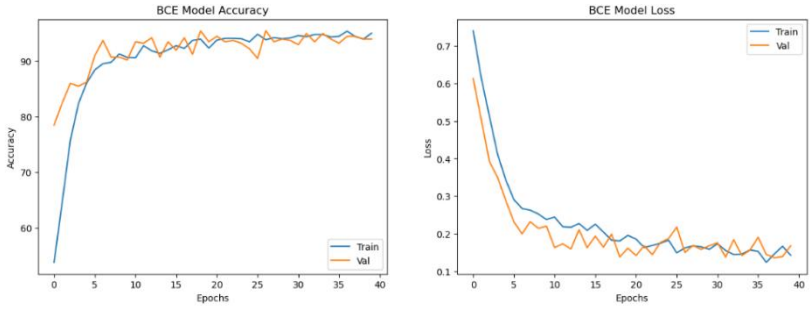

Fig14.使用 LeakyReLU 函數的 model 之表現

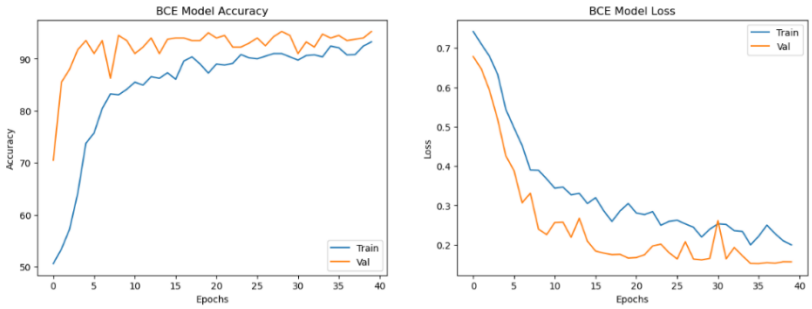

Fig15.使用 ReLU 函數的 model 之表現